Mitigating OWASP API security top 5 risks through API gateway patterns

Liviu MOISEI,
Technical University of Moldova, Faculty of Computers Informatics and Microelectronics, Information Security, Chisinau, Republic of Moldova
liviu.moisei@isa.utm.md

Liviu MOCANU,
Technical University of Moldova, Faculty of Computers Informatics and Microelectronics, Information Security, Chisinau, Republic of Moldova
liviu.mocanu@isa.utm.md

Abstract

Objectives: This study aims to analyze how API gateway patterns can mitigate the top five risks identified in the OWASP API Security Top 10. Given the critical role of APIs in modern applications, addressing these vulnerabilities is essential for maintaining robust security postures. **Prior Work**: Building upon existing literature and industry reports, this paper examines the evolution of API security threats, particularly the updates in the 2023 OWASP list, and the corresponding mitigation strategies employed through API gateways. **Approach**: The research involves a comprehensive review of academic sources, industry publications, and real-world case studies to evaluate the effectiveness of API gateway patterns – such as rate limiting, authentication and authorization, schema validation, and logging – in mitigating identified security risks. **Results**: The analysis reveals that implementing these gateway patterns significantly enhances API security. For example, rate limiting effectively prevents DoS attacks by controlling request rates, while robust authentication and authorization mechanisms ensure that only legitimate users access sensitive endpoints. **Implications**: The findings provide actionable insights for security architects and developers, emphasizing the importance of integrating API gateway patterns with other security measures, such as secure coding practices and runtime protection, to create a layered defense strategy. **Value**: This paper bridges the gap between theoretical security frameworks and practical implementation, offering a roadmap for leveraging API gateway technologies as a frontline defense in API security.

Keywords: access control, payload validation, web security, microservices, rate limiting.

1. Introduction

APIs (Application Programming Interfaces) have become foundational to digital transformation, enabling organizations to build scalable, modular, and interoperable systems. With the growing adoption of microservices and cloud-native architectures, the number of publicly and privately exposed APIs has surged, bringing with it a corresponding increase in API-specific security threats.

Recognizing this, the Open Web Application Security Project (OWASP) introduced the API Security Top 10, a curated list of the most critical security issues affecting APIs. These include familiar concerns like broken authentication and injection, as well as API-specific challenges such as mass assignment and insufficient logging. One promising avenue for mitigating these risks lies in the API gateway – a centralized component that manages, routes, and secures API traffic. Beyond basic request routing, modern API gateways offer a suite of security-enhancing features such as rate limiting, access control, payload validation, and threat detection [1]. This review explores how various API gateway patterns can be leveraged to address 5 of the OWASP API Security Top 10 threats. By synthesizing research findings and industry practices, the paper aims to provide a practical and strategic framework for API security, focusing on the proactive role that gateway-level controls can play in a layered defense approach.

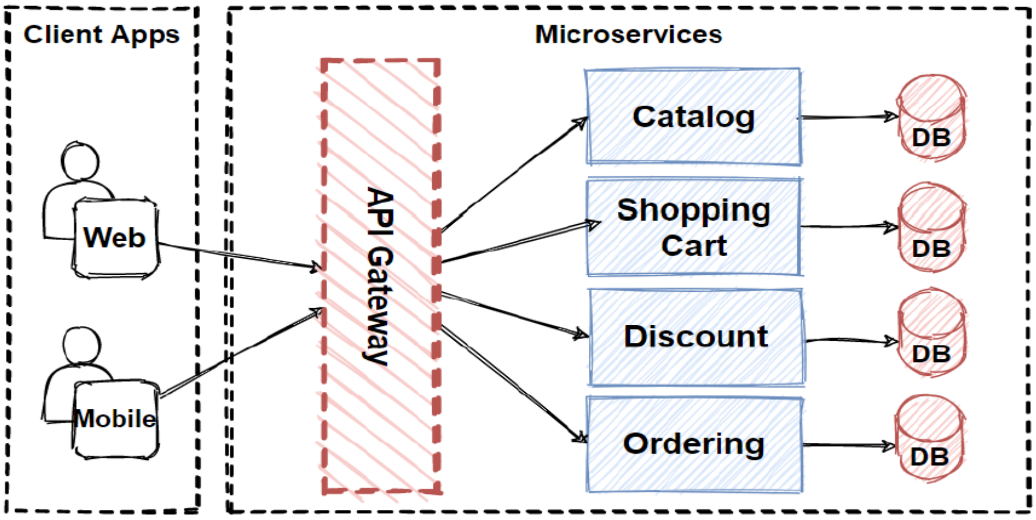

Fig. 1. API Gateway as a centralized enforcement layer between client applications and microservices
Source: "API Gateway Pattern" by M. Ozkaya, 2021, Medium

2. Overview of OWASP API security

The OWASP API Security Top 10 is a project developed by the Open Web Application Security Project to identify and prioritize the most critical API-related security threats. These threats are based on real-world incidents and input from the industry. In this section, we present top 5 risks that are effectively mitigated by API gateway patterns and analyze how well the pattern can help mitigate it, referencing best practices.

2.1. API1:2023 – Broken object level authorization

Broken Object Level Authorization (BOLA) is one of the most common and dangerous API vulnerabilities [2]. It occurs when an API exposes endpoints that accept user-supplied object identifiers (such as account IDs or document keys) but fails to enforce proper authorization checks on those objects. This allows attackers to manipulate IDs and gain access to resources they don’t own, leading to data breaches or unauthorized actions.

API gateways play a crucial role in defending against BOLA by acting as a first line of defense:

1. Request Validation and Schema Enforcement. API gateways can enforce strict schemas for incoming requests, ensuring only expected formats and fields are processed. Tools like Kong and AWS API Gateway support JSON schema validation, preventing attackers from submitting malformed or unauthorized object IDs.
2. Authorization Enforcement: API gateways can integrate with external policy engines (e.g., OPA) or IAM systems to enforce object-level access policies. For example, the gateway can call out to a policy service that verifies if a user is allowed to access a given

object before forwarding the request. Some platforms allow declarative access control using RBAC or ABAC models.
3. Context-Aware Policies: Modern gateways can inspect JWT claims or user context and apply route-level policies accordingly. For example, if a user tries to access /orders/123, the gateway can verify if user_id in the JWT matches the owner of order 123. This enables fine-grained, attribute-aware control.

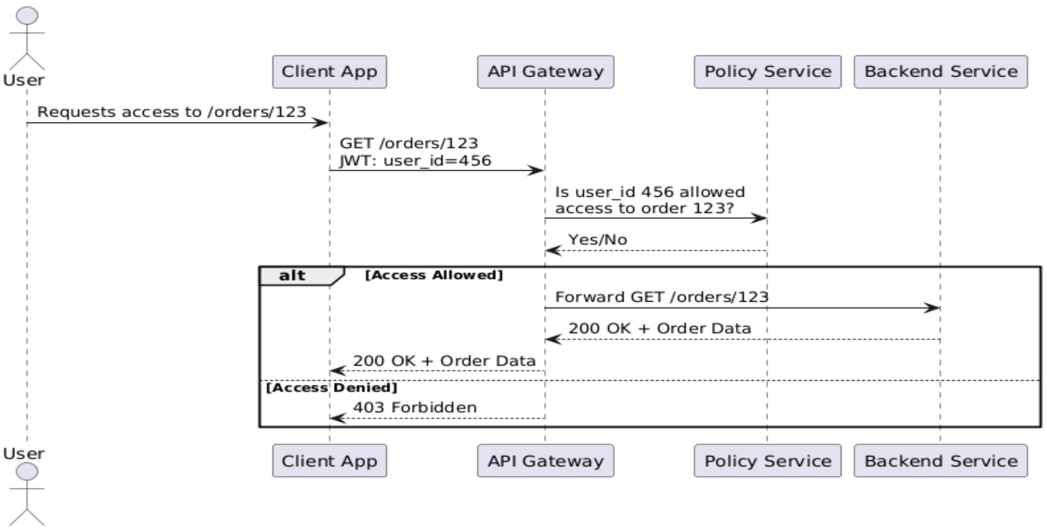

Fig. 2. Gateway-mediated mitigation of Broken Object Level Authorization through policy evaluation
Source: Own work.

Whilst gateways are effective at enforcing high-level policies, they may lack the internal business context required for true object-level access validation. Therefore, a layered approach is recommended where gateways filter obvious violations, while backend services perform deeper authorization checks.

2.2. API2:2023 – Broken authentication

Broken Authentication refers to failures in correctly implementing authentication mechanisms, which can lead to unauthorized access [3]. This includes insecure token storage, lack of multi-factor authentication (MFA), weak password enforcement, token replay vulnerabilities, and inconsistent session handling. APIs relying solely on session tokens or poorly implemented authentication flows are vulnerable to impersonation and account takeover attacks.

1. Centralized Authentication Enforcement. API gateways provide centralized enforcement of authentication using modern protocols like OAuth2, OpenID Connect, and SAML. This ensures uniform application of authentication policies across microservices and minimizes inconsistencies.
2. JWT and Token Validation. Gateways such as Apigee, AWS API Gateway, and Kong can inspect and validate JSON Web Tokens (JWTs) by verifying their signature, expiration, and issuer. This prevents unauthorized access using expired, forged, or tampered tokens.

3. Support for Multi-Factor Authentication (MFA). Gateways can integrate with identity providers (IdPs) to enforce MFA requirements before forwarding requests to sensitive routes. This is particularly important for operations involving financial transactions or access to personal data.
4. Replay Attack Prevention. Advanced gateway configurations allow detection of token reuse and replay attacks by checking request timestamps, nonces, or sequence numbers – often in conjunction with backend session management.

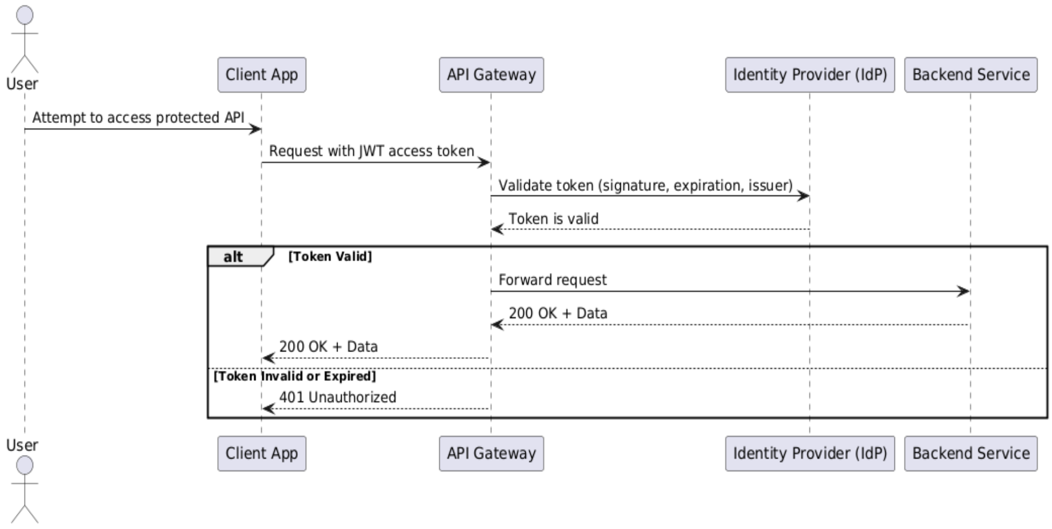

Fig. 3. Authentication enforcement via Gateway using token validation and identity provider integration
Source: Own work.

Gateways can enforce authentication on the perimeter, but internal services must still handle fine-grained access control and session context securely. Additionally, gateway-level authentication should be complemented with transport-layer security (TLS), secure token storage on clients, and robust user verification mechanisms.

2.3. API3:2023 – Broken object property level authorization

Broken Object Property Level Authorization occurs when users are authorized to access an object but can manipulate fields within that object that they should not be able to read or modify. This flaw is particularly common in APIs exposing complex JSON payloads where sensitive attributes like internal roles, permissions, pricing, or user flags – are insufficiently protected [4].

1. Response Transformation and Filtering. Modern API gateways (e.g., Apigee, Kong) offer plugins and policies that dynamically modify or redact specific fields from responses based on the requesting user's identity. This reduces the likelihood of unintentionally exposing sensitive fields to unauthorized users.
2. Field-Level Access Control. Advanced gateways with policy-as-code support (like OPA or WSO2) allow defining granular access policies at the property level. For instance, admins might retrieve full user profiles, while customers receive filtered views excluding internal-only fields.

3. Role-Aware Schema Validation. API schema enforcement can be enhanced with conditional logic, applying different schemas based on the role or group of the requester. Gateways can bind a particular schema to a JWT claim or API key scope, filtering both inbound and outbound data structures.

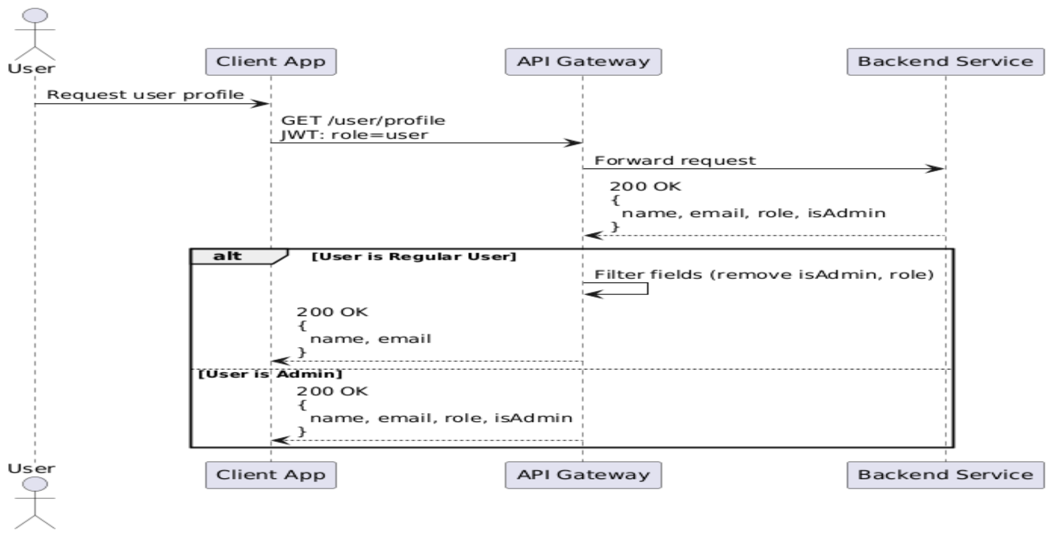

Fig. 4. Gateway-mediated mitigation of Broken Object Level Authorization through policy evaluation
Source: Own work.

Even though gateways are effective for response shaping, they do not inherently understand deep domain-specific logic. Field-level policies must complement internal backend logic, and care must be taken not to rely solely on gateway masking. Additionally, runtime transformations may introduce performance costs at scale.

2.4. API4:2023 – Unrestricted resource consumption
This vulnerability arises when APIs allow clients to consume excessive resources such as CPU, memory, bandwidth, or database connections – without proper limits [5]. Unlike traditional DoS attacks that target infrastructure, API-level resource exhaustion can be subtle and exploit logical abuse like excessive pagination, large file uploads, or frequent polling.

1. Rate Limiting and Quotas. One of the most effective defenses against resource consumption is enforcing request limits. API gateways like Kong, Apigee, and AWS API Gateway support IP-based, user-based, or key-based rate limiting and quota enforcement. These policies throttle abusive clients before requests hit the backend.
2. Request Size Limiting. Gateways can enforce maximum payload sizes for both requests and responses. This helps prevent attackers from sending massive JSON payloads or uploading large files that could overwhelm the system.
3. Pagination Limits and Input Constraints: Some gateways allow validation of query parameters like limit, page, or offset. You can define constraints (e.g., limit <= 100) to prevent inefficient queries or data scraping attacks.

4. Concurrent Request Control. Certain gateway platforms support concurrency control policies that restrict how many simultaneous requests a client can make. This prevents flooding scenarios in multi-threaded or async environments.
5. Caching and Response Optimization. By enabling response caching at the gateway level for non-sensitive GET requests, platforms can offload frequently accessed data and reduce backend strain – particularly useful for APIs that return static or slow-changing data.

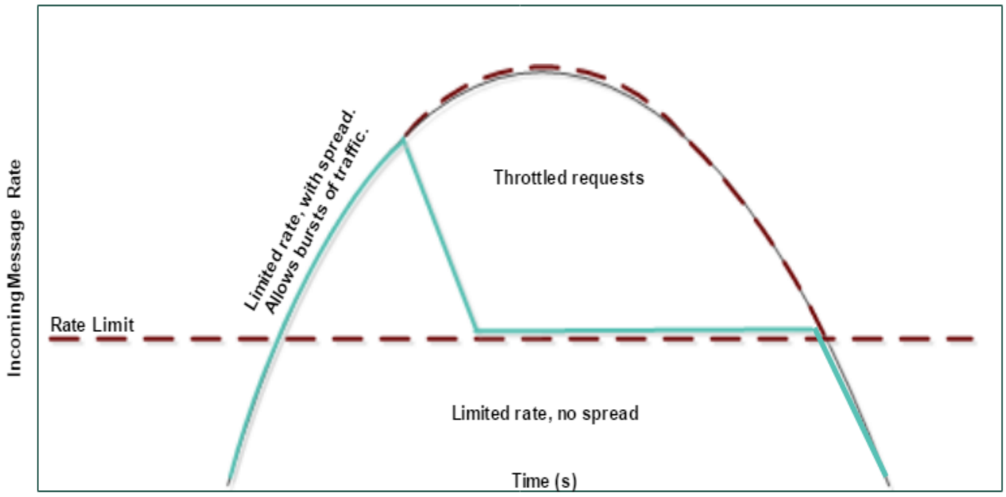

Fig. 5. Rate limiting behaviour enforced by the Gateway showing throttled requests exceeding the limit
Source: "Apply Rate Limit Assertion" by Broadcom, 2024

Whilst API gateways can mitigate many forms of overuse, they can not detect business logic abuse without context. For instance, repeated use of a costly but valid function (e.g., dynamic PDF generation) may bypass technical rate limits but still cause load issues. Backend services should monitor usage patterns and integrate behavioral analytics or usage thresholds to complement gateway-level protections.

2.5. API5:2023 – Broken function level authorization

This vulnerability occurs when an API fails to restrict access to sensitive functions or operations based on user roles or privileges [6]. While object-level authorization (API1) protects specific data, function-level authorization ensures that only appropriate users can invoke specific operations, such as administrative actions or data deletions.

1. Route-Based Access Policies. API gateways allow defining fine-grained access policies at the route or method level. For instance, requests to /admin/* can be restricted to users whose JWT includes a role=admin claim. This prevents unauthorized access before the request reaches the backend.
2. Role-Based Access Control (RBAC) Integration. Gateways can enforce RBAC policies using external identity providers (e.g., Keycloak, Auth0) or integrate with centralized policy engines like Open Policy Agent. These tools evaluate the user's roles or groups against route-level access rules defined in the gateway configuration.

3. Scope Validation and API Key Permissions. In OAuth2 flows, gateways can inspect access tokens for specific scopes (e.g., read:users, write:settings) and block requests that lack required permissions. Similarly, API key-based access can be tied to allowed operations via API plans or service tiers.
4. HTTP Method Filtering. Gateways can restrict HTTP methods per route. For example, only GET may be allowed on /user/account for regular users, while DELETE or PATCH is permitted only for admins. This reduces the attack surface of potentially destructive operations.

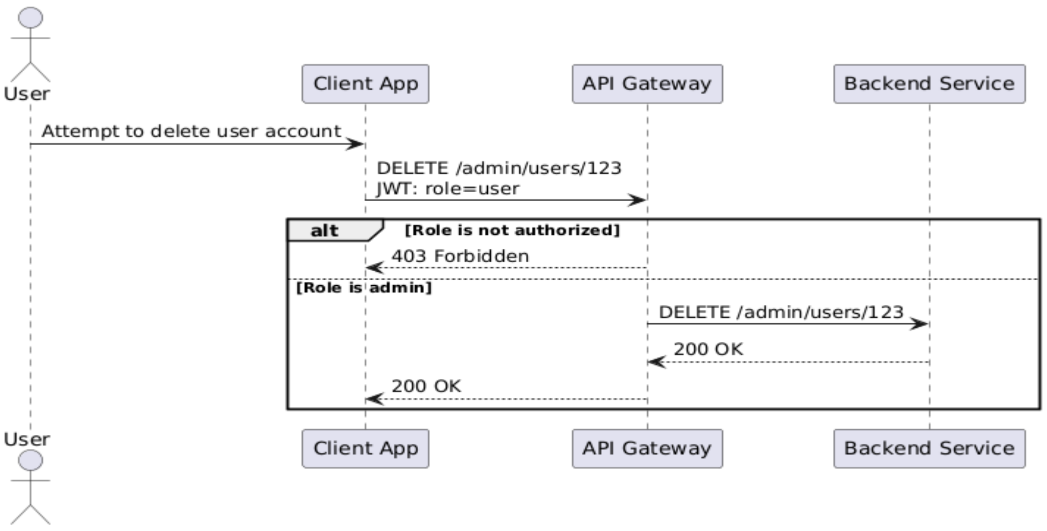

Fig. 6. Role-based route and method authorization enforced by Gateway to block unauthorized access
Source: Own work.

Despite the fact that gateways can prevent unauthorized function calls on the perimeter, internal services must still validate user roles and permissions to prevent privilege escalation or bypass through internal APIs or service chaining. Moreover, defining and maintaining access rules for every function or route at the gateway level can become complex at scale.

3. Conclusion

API gateways play a crucial role in mitigating the top five OWASP API Security risks by enabling centralized enforcement of access control, rate limiting, schema validation, and token-based authentication. As APIs become central to modern applications, these gateway patterns provide consistent, scalable protection across services.

The analysis reveals that API gateways are particularly effective for addressing perimeter security concerns like broken authentication (API2) and unrestricted resource consumption (API4), where centralized policy enforcement can block malicious requests before they reach backend services. For object-level authorization issues (API1 and API3), gateways provide a valuable first line of defense but must be complemented by service-level validation that understands the business context of data relationships. Function-level authorization (API5) benefits significantly from gateway-enforced route and method restrictions tied to user roles and scopes.

While gateways offer strong perimeter defenses, they are not sufficient alone. True resilience requires layering gateway protections with secure coding practices, thorough backend validation, and proper identity management. This multi-layered strategy is essential because certain vulnerabilities, particularly those involving complex business logic or data relationships, may bypass gateway-level controls.

Future research should explore how emerging technologies like artificial intelligence and machine learning can enhance API gateway security through anomaly detection and adaptive policy enforcement [7, 8, 9]. Additionally, as API ecosystems grow more complex, further investigation into scalable, automatable security patterns will be essential for maintaining effective protection across thousands of endpoints and services.

References

[1] W. Joseph and A. Jones, "Modern API Security Frameworks for Enterprise Systems," 2025.

[2] J. A. Diaz-Rojas, J. O. Ocharán-Hernández, J. C. Pérez-Arriaga and X. Limón, "Web API Security Vulnerabilities and Mitigation Mechanisms: A Systematic Mapping Study," in *9th International Conference in Software Engineering Research and Innovation (CONISOFT)*, 2021.

[3] H. Adebayo, "Digital Banking and API Security: Best Practices for Secure Financial Transactions," 2025.

[4] C. Zhao, "API Common Security Threats and Security Protection Strategies," *Frontiers in Computing and Intelligent Systems*, vol. 10, no. 2, 2024.

[5] P. Dixit, "How to Secure APIs to Defend Against Emerging Cyber Threats to Digital Web Assets," *International Journal of Computer Trends and Technology*, vol. 72, no. 9, 2024.

[6] A. A. Kumar and T. L. Divya, "Security measures implemented in RESTful API Development," *Open Access Research Journal of Engineering and Technology*, vol. 7, no. 1, 2024.

[7] J. Paul, "Building a Robust API Security Framework with Machine Learning," 2024.

[8] J. Paul, "Integrating Machine Learning with API Gateway Security Solutions," 2024.

[9] D. Kaul and R. Khurana, "AI to Detect and Mitigate Security Vulnerabilities in APIs: Encryption, Authentication, and Anomaly Detection in Enterprise-Level Distributed Systems," 2021.